Paul Singman
12-18-2015
Udacity MLE Project # 2

Supervised Learning – Building an Intervention System

## Classification vs Regression

The goal of this project is to identify students who need early intervention so that as many students as possible can pass final exams and graduate. This is a classification problem since we are trying to classify students into two categories—either needing or not needing intervention—based on the their predicted probability of passing the exam. If we were trying to predict a continuous variable, like a student's score on the exam for example, then it would be a regression problem.

## Exploring the Data

```
Total number of students: 395
Number of students who passed: 265
Number of students who failed: 130
Number of features: 30
Graduation rate of the class: 67.09%
```

## Preparing the Data

The code to identify feature and target columns, preprocess feature data, and split data into training and testing sets was executed. Of the 395 data points, 300 were random selected to be part of the training set, and the remaining 95 were reserved for the testing set.

## Training and Evaluation Models

Three supervised learning models were fitted to the data that are appropriate for a classification task. The models chosen were:

- Logistic Regression
- Support Vector Machine
- Decision Tree Classifier

## Logistic Regression

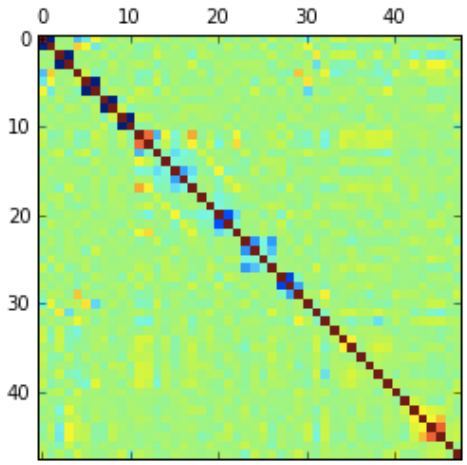
The O(n) complexity of the logistic regression model in terms of input size is $O(np^2)$, where n is the number of observations and p is the number of features. This means the model is computationally more sensitive to additional numbers of features than observations.

Logistic regression is generally applied to classification tasks, where the dependent variable being predicted is qualitative. The output of a logistic regression model is a value between 0 and 1, which can be interpreted as the probability that an observation belongs to a class. Although any threshold is acceptable, it is common to say that if p(X) > 0.5, then X can be classified into category 1, and vice versa.

One of logistic regression's strengths is that it is conceptually relatively straightforward as it is part of the family of algorithms known as Generalized Linear Models. To be specific, the feature parameters it solves for are easily interpretable, as they are linear to the logged odds ratio of p(X). Logistic regression also benefits from making relatively few assumptions about the underlying data; specifically it does not require that the independent variables, nor the residual errors, be normally distributed.

A weakness of logistic regression is that it has limited explanatory power compared to some of the more modern algorithms like neural networks or ensemble methods. It also requires a large number of training examples per feature. Rules of thumb suggest 10-30 per feature. Another weakness is the independent variables should be independent of each other (low multicollinearity) or else the model loses some of its interpretability with respect to individual predictors.

With that said, let's explore the multicollinearity of the features using a correlation matrix:




From this plot we can see obviously the features are perfectly correlated with themselves as shown by the dark red line down the diagonal of the plot. In addition, the features that were created by the get_dummies function are perfectly negatively correlated, so it will be important to exclude one of them from each set when fitting the model. None of the other features appear to be particularly correlated with another so I will include all of the other features.

Excluding one of the dummy variables from each set leaves 39 features on which to train the model. I chose a logistic regression model model because I believe it is a good baseline model to fit when doing a classification task even though I wouldn't expect it to outperform more complicated models that will be fit later. However, I do expect it to use less computational resources than other models.

Here is a summary table of the performance of the logistic regression model:

**Logistic Regression Results Summary**

| | Training Set Size | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Training Time | 0.002 | 0.005 | 0.005 |
| Prediction Time | 0.00… | 0.00… | 0.00… |
| F1 Score Train Set | 0.877 | 0.803 | 0.777 |
| F1 Score Test Set | 0.740 | 0.681 | 0.751 |

The model took fractions of a second to train and make predictions off the training data. With all of the training data, the logistic regression model produced an F1 score of 0.751. Let's see what another model can do.

**Support Vector Machine**

The next type of model that will be tested is a support vector machine. Although the O(n) time of an SVM can vary based on the kernel function used, in general support vector machines are an $O(n^2p)$ algorithm. This means the model is computationally more sensitive to additional numbers of observations than features.

I chose this model because SVMs are effective at classification in high dimensional spaces (i.e. when there are many features) and when a non-linear decision boundary is needed. Also SVM's do not require multiple observations for each feature, which is important for this dataset with less than 10 observations for each feature. SVMs are also not sensitive to outliers since the only points that affect the optimal solution are the ones closest to the hyperplane it solves for.

A downside of an SVM is it does not provide probability estimates for each observation. It is also more difficult to explain the workings of the model and interpret the results.

Here is a summary table of the performance of the support vector machine model:

**Support Vector Machine Results Summary**

| | Training Set Size | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Training Time | 0.012 | 0.006 | 0.009 |
| Prediction Time | 0.002 | 0.003 | 0.006 |
| F1 Score Train Set | 0.849 | 0.819 | 0.815 |
| F1 Score Test Set | 0.841 | 0.827 | 0.827 |

Although the SVM takes slightly longer to train and predict than logistic regression, the time is still fractions of a second. Compared to logistic regression, the SVM predicted students better, with an F1 score of about 0.83.

**Decision Tree Classifier**
The final model to be tested is a decision tree classifier. Decision trees performance is O(nlog(n)p), which assumes the tree is relatively balanced.

Decision trees benefit from being easy to understand and interpret. They also can be fit on almost any type of dataset, even one with a mix of numerical & categorical data, and data that is unscaled.

Decision trees are notorious for overfitting datasets without parameters in place to prevent overfitting, like setting a max depth for the tree, or a minimum number of samples at each node. Decision tree algorithms are greedy, meaning they won't always build the optimal tree. This issue can mostly be negated by building multiple trees, though this adds additional computational cost.

Here is a summary table of the performance of the decision tree model:

**Decision Tree Classifier Results Summary**

| | Training Set Size | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Training Time | 0.006 | 0.001 | 0.003 |
| Prediction Time | 0.00… | 0.00… | 0.00… |
| F1 Score Train Set | 1.0 | 1.0 | 1.0 |
| F1 Score Test Set | 0.710 | 0.640 | 0.625 |

Although the decision tree built was able to perform perfectly on the training data, it did not generalize well on the test data.

**Choosing the Best Model**

Overall the model with the best F1 scores is the support vector machine, with an F1 score of 0.827 on whole training dataset compared to logistic regression's 0.751 and the decision tree's 0.625. Even though the SVM is also the most computationally expensive algorithm, I still recommend it as the optimal model. For the SVM, training time is still just fractions of a second, and I don't consider this to be a time-sensitive project (compared to, say, a recommender system that needs to generate real-time recommendations to users) nor is the dataset likely to ever grow so large where maximizing computational resources becomes relatively more important than accuracy.

**Describing the Model in Layman's Terms**

The Support Vector Machine is an algorithm that solves for the best line that correctly separates the data points so that all of the points in one class (or group) are on one side of the line, and all of the points in the other class are on the other side of the line. The 'best' in the phrase 'best line' refers to the line that can be moved the most in either direction and still make the correct classifications. Using a more technical term, it has the largest margin. Although this description describes a SVM with two-dimensional data, the algorithm can be extended to work in higher dimensions, where instead of solving for the best line to separate the data into classes, it solves for the best 'hyperplane'. Predicting the class of new examples is as simple as calculating which side of the line (or hyperplane) the new point falls on.

**Model Tuning**

I will now further tune the model with gridsearch by adjusting the C parameter. This parameter is the penalty parameter on the error term and is analogous to regularization. A large C means a low penalty, and a small C means there is a higher penalty. I'll test a few values of C on the scale from .01 to 1,000 to see which produces the best results.

The results from the gridsearch are shown below:

```
In [38]:grid_search.grid_scores_
Out[38]:
[mean: 0.78788, std: 0.00000, params: {'C': 0.1},
 mean: 0.78611, std: 0.00621, params: {'C': 1},
 mean: 0.65956, std: 0.02046, params: {'C': 10},
 mean: 0.63907, std: 0.02152, params: {'C': 1000}]
```

As you can see, the SVM fit with a C parameter of 0.1 produced the best F1 Score (labeled as 'mean' in the output) of 0.78788. This is actually slightly lower than the F1 Scores of the SVM fit earlier which were approximately 0.83 and uses the Sklearn default of C=1. I attribute this to random sampling of the data, and consider the gridsearch model approximately as good as the original.